energetics
technology center

Developing Autonomy for Unmanned Surface Vehicles by Using Virtual Environments

Principal Investigator:
Robert Kavetsky

Authors:
Robert Kavetsky, Energetics Technology Center
Satyandra K. Gupta, PhD, University of Maryland

20110112391

REPORT DOCUMENTATION PAGE				Form Approved OM8 No. 0704-0188

The public reporting burden for this collection of information is astimated to average 1 hour par response, including tha tima for reviewing instructions, saarching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defensa, Washington Headquartars Services, Directorata for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202 4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY) 11/10/2010	2. REPORT TYPE FINAL TECHNICAL REPORT		3. DATES COVERED (From - To) 6/27/2008--11/10/2010
4. TITLE AND SUBTITLE DEVELOPING AUTONOMY FOR USVs BY USING VIRTUAL ENVIRONMENTS		**5a. CONTRACT NUMBER** N/A	
		5b. GRANT NUMBER N00014-08-1-1108	
		5c. PROGRAM ELEMENT NUMBER N/A	
6. AUTHOR(S) ROBERT KAVETSKY SATYANDRA K. GUPTA		**5d. PROJECT NUMBER** N/A	
		5e. TASK NUMBER N/A	
		5f. WORK UNIT NUMBER N/A	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) ENERGETICS TECHNOLOGY CENTER, INCORPORATED 107 CENTENNIAL STREET, SUITE 201 LA PLATA, MARYLAND 20646		**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) OFFICE OF NAVAL RESEARCH 875 NORTH RANDOLPH STREET ARLINGTON, VIRGINIA 22203-1995		**10. SPONSOR/MONITOR'S ACRONYM(S)** N/A	
		11. SPONSOR/MONITOR'S REPORT NUMBER(S) N/A	

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

13. SUPPLEMENTARY NOTES

N/A

14. ABSTRACT

We address the problem of automated synthesis of an action selection policy for an unmanned vehicle operating in an environment with a deceptive adversary. We introduce a new synthesis approach using which an initial version of the policy is automatically generated and then gradually refined by detecting and fixing its shortcomings.

Our focus is specifically on automated synthesis of a policy used for blocking the advancement of an intruder boat toward a valuable target. The USV must generate specific maneuvers for blocking. The intruder is human competitive and exhibits a deceptive behavior so that the USV cannot exploit regularity in its attacking behavior.

We compared the performance of a hand coded USV's blocking policy to the performance of a policy that was automatically synthesized. Our results show that the performance of the automatically generated USV's policy exceed the performance of the hand coded policy and thus demonstrates the feasibility of the proposed approach.

15. SUBJECT TERMS

UNMANNED SURFACE VEHICLE, BLOCKING ACTIONS, AUTOMATED SOFTWARE GENERATION, AUTONOMOUS BEHAVIOR

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON ROBERT KAVETSKY
a. REPORT	b. ABSTRACT	c. THIS PAGE	SAR	42	
NONE	NONE	NONE			19b. TELEPHONE NUMBER (Include area code) (301) 934-0385

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Developing Autonomy for USVs by Using Virtual Environments

Abstract

We address the problem of automated synthesis of an action se-
lection policy for an unmanned vehicle operating in an environment
with a deceptive adversary. We introduce a new synthesis approach
using which an initial version of the policy is automatically generated
and then gradually refined by detecting and fixing its shortcomings.
The synthesis mechanism consists of the automated extraction of the
vehicle's states of exception and Genetic Programming (GP) for au-
tomated generation of corrective sequences of action commands to be
applied locally.

Our focus is specifically on automated synthesis of a policy used
for blocking the advancement of an intruder boat toward a valuable
target. This task requires the USV to utilize reactive planning com-
plemented by short-term forward planning to generate specific ma-
neuvers for blocking. The intruder is human competitive and exhibits
a deceptive behavior so that the USV cannot exploit regularity in its
attacking behavior.

We compared the performance of a hand coded USV's blocking pol-
icy to the performance of a policy that was automatically synthesized.
Our results show that the performance of the automatically generated
USV's policy exceeds the performance of the hand coded policy and
thus demonstrates the feasibility of the proposed approach.

1 Introduction

Manual development of a truly robust unmanned robotic vehicle for tasks in
which the unmanned system has to autonomously face a human-competitive

1

adversary exhibiting deceptive behavior is a challenge. This scenario is typical for a combat mission task where even a single mistake in the decision of the vehicle can have fatal consequences. In such scenarios, the unmanned vehicle has to be prepared to rapidly execute specialized maneuvers to its default strategy in specific situations as defined by its control algorithm, or policy to successfully accomplish its task. The difficulty in the development of such a policy consist in manual handling of the vehicle's failure states that arise in the encountered environments. This requires intensive repeatable testing of the overall vehicle's behavior using a large suite of different test scenarios, identifying the shortcomings, and implementing various contingency-handling behaviors (Baker et al 2008).

In this report, we present a new approach for automated policy synthesis for unmanned vehicles operating in a continuous state-action space. This approach can be viewed as a synthesis process during which an initial version of the policy is automatically generated and then gradually improved by detecting and fixing its shortcomings. Each iteration of this process involves extensive evaluation of the overall vehicle's behavior in multiple simulation runs, automatically finding states of failure in which the vehicle exhibited deficiencies in accomplishing its task, selecting a state from which the vehicle can avoid the largest amount of the detected faulty states by executing an appropriate maneuver, and letting the vehicle to learn an improved maneuver. This way, the focus of the synthesis process is always on the most crucial encountered states of failure. At the end of the policy synthesis, the vehicle thus knows which additional maneuvers to execute in which states to enhance the performance of its default policy.

The synthesis mechanism is completely automated and consists of the extraction of the vehicle's states of exception and Genetic Programming (GP) (Koza 2003) for generation of corrective sequences of action commands to be applied locally. During the whole process, no external information on how the policy should be synthesized is therefore needed. The policy is internally represented as a composite consisting of one default high-level controller and a set of specialized action plans. The default controller is used to control the vehicle's behavior in all situations besides the situations for which specific maneuvers represented as action plans are needed. The action space of the vehicle is represented by a set of primitive commands, each having continuous parameters. The commands are combined, parametrized, and composed into a structure by the synthesis process to perform the overall task. The inherent symbolic representation of the policy greatly simplifies the analysis of its
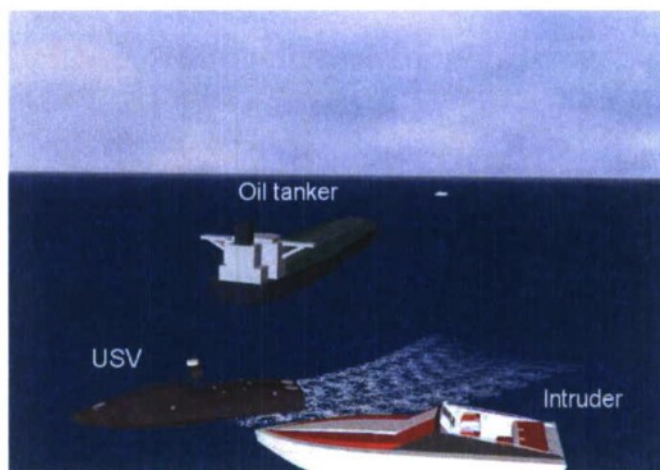
Figure 1: Virtual environment: Unmanned Surface Vehicle (USV) is protecting an oil tanker by blocking the advancement of an intruder boat

behavior. Moreover, the symbolic representation allows integrating human knowledge and the analysis of the policy can provide the basis for improving the code.

Our approach was tested in the context of a large project aimed at the development of a general mission planning system (Schwartz et al 2009; Svec et al 2010) for the automated synthesis of action selection policies for Unmanned Surface Vehicles (USVs) (Corfield and Young 2006; Finn and Scheding 2010). In this report, we extend our previous work (Svec and Gupta 2010) in which the focus is specifically on automated synthesis of a policy used for blocking the advancement of an intruder boat toward a valuable target (see Fig. 1). This task requires the USV to utilize reactive planning complemented by short-term forward planning to generate local action plans describing specific maneuvers for the USV. The intruder is human-competitive in the sense that its attacking efficiency approaches the attacking efficiency of deceptive strategies exhibited by human operators. Our aim is to reach the level 3 of autonomy as defined in (Board 2005). In this level, the unmanned vehicle automatically executes mission-related commands when response times are too short for operator intervention.

An overview of the overall approach is shown in Fig. 2. First, we de-

veloped a physics-based meta-model using a detailed dynamics model of the USV to be able to test the policy in a simulation environment in real-time (Thakur and Gupta 2010). Second, we developed a mission planning system that contains a policy synthesis module (see Section 5). The necessary system architecture of the USV including the policy and state representation is described in Section 3. In order to combine the elements of the project into a cohesive system, we designed a USV simulation environment (see Section 4) (Svec et al 2010). The USV simulation environment integrates various components of the project into a complete simulation system and acts as a simulation platform for the synthesis module. One of the components of the USV simulation environment is the virtual environment (VE) based simulator (see Fig. 1) which serves as an emulator of the real USV environment that allows human players to play against each other or against the computer. Finally, we present an experimental evaluation of the approach in a challenging simulated combat-like scenario to demonstrate the feasibility of the proposed approach (see Section 6).
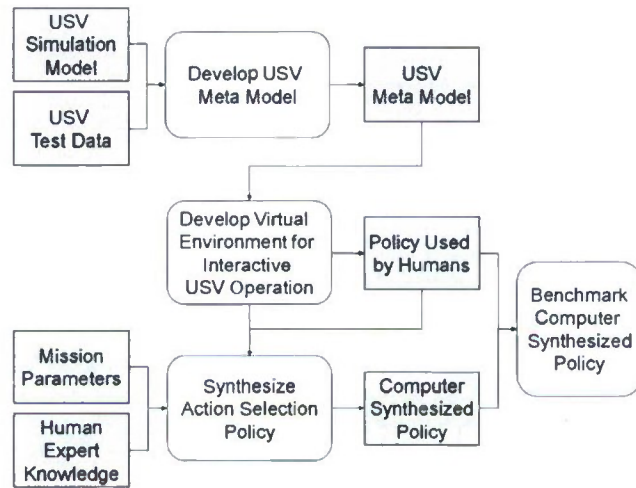

Figure 2: Overview of the overall approach

2   Related Work

Computational synthesis (Lipson et al 2003) deals with the problem of how to automatically compose and parametrize a set of functional building blocks into a hierarchical solution structure with the desired functionality. This is in contrast to classical optimization in which the number and structure of modules and parameters being optimized is known in advance.

Evolutionary Robotics (ER) (Floreano and Mattiussi 2008) is a methodology that uses evolutionary algorithms to automatically synthesize controllers and body configuration for autonomous robots. As opposed to the use of standard temporal difference methods to approximate a value function (Sutton and Barto 1998), simulated evolution searches directly for a policy that maps states to actions.

In literature, there are many successful applications of evolutionary techniques to robots' controllers synthesis. In many cases, the representation of the evolved controllers is a neural network. In the domain of neuroevolution, a popular method is the Neuroevolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen 2002) that was successfully applied to many controller synthesis problems. The main issue with the neural network representation, however, is the difficulty of analyzing the evolved solutions.

Besides the possibility to use methods for rules extraction from the evolved neural networks (Diederich et al 2010), controllers can also be directly synthesized in a symbolic form. One of the techniques used to generate symbolic controllers is Genetic Programming (GP) (Koza 2003). GP as one of the robust evolutionary techniques has been used for automatically generating computer programs that usually have a tree structure and are generated using an algorithm similar to the traditional genetic algorithm (GA) (Goldberg 1989). Most of the controllers were successfully evolved for a wide variety of behaviors as obstacle avoidance (Barate and Manzanera 2007; Nehmzow 2002), wall-following (Dain 1998), line following (Dupuis and Parizeau 2006), light seeking, robot seeking (Nehmzow 2002), box pushing (Koza and Rice 1992), vision-driven navigation (Gajda and Krawiec 2008), homing and circling (Barlow and Oh 2008), predator versus prey strategies (Haynes and Sen 1996), co-evolution of control and bodies morphologies (Buason et al 2005), game playing (Jaskowski et al 2008; Togelius et al 2007; Doherty and ORiordan 2006) or group control for survey missions (Richards et al 2005). GP was also utilized for the automated synthesis of human-competitive strategies for robotic tanks run in a closed simulation area to fight other human-designed

tanks in international leagues (Shichel et al 2005). There was also some progress on development of limited machine intelligence for classical strategic games like backgammon or chess endgames (Sipper et al 2007).

However, for high-level strategy problems, e.g. the keep-away soccer (Kohl and Miikkulainen 2008), the discontinuous structure of the state-action space prohibits the standard evolutionary algorithms to generate good solutions. The monolithicity and the reactive behavior of the evolved controllers limit their usage for complex fractured domains (Kohl and Miikkulainen 2008) in which the robot's best actions can radically change as the robot is moving continuously from one state to the other.

In the domain of symbolic controllers, Learning classifier system (LCS) (Bacardit et al 2008, Lanzi 2008, Ryan et al 2009) can be considered as one of the best options to cope with the fractured state-action space problems in robot learning. LCS represents a comprehensive class of evolutionary systems that solve reinforcement learning problems by evolving a set of classifiers or rules responsible for handling different local parts of input spaces of the problems. Our proposed approach extends the general paradigm of these systems to highly dynamic and continuous environments. In LCS, the genetic algorithm searches for an adequate decomposition of the problem into a set of subproblems by evolving classifier conditions. In our approach, we iteratively detect states in which the robot exhibits deficiencies in fulfilling its task and then find new states from which synthesized specialized action plans can be executed to avoid the failure states. We use GP as a discovery component for the action part of the classifiers.

In the neuroevolution domain, there are examples of methods used for synthesizing different behaviors for different circumstances (so called multi-modal behaviors (Schrum and Miikkulainen 2009) arising in continuous state-action spaces. However, it is difficult to learn such behaviors reliably and without extensive human intervention, e.g. as is the case of the layered evolution combined with the subsumption architecture (van Hoorn et al 2009). Moreover, the number of behaviors to evolve is usually determined in advance by a human designer. In most applications, the behaviors are learned and the control mechanism is hand coded, or the control mechanism is learned and the behaviors are hand-coded. In some cases, both behaviors and the control mechanism are learned but separately and with extensive human help.

Recently, neuroevolutionary methods have been developed that discover multi-modal behaviors automatically and do not depend on knowledge of the task hierarchy. For example, a special mutation operator was proposed

6

in (Schrum and Miikkulainen 2009) for evolving special output nodes that control the mode of the behavior. However, the potential number of modes needed to be evolved can be very high for some tasks. Another work (Kohl and Miikkulainen 2008) extends NEAT to use radial basis function (RBF) (Buhmann 2001) nodes to evolve controllers with better performance for complex problems. Similarly, HyperNEAT has been developed as another extension of the NEAT method with the ability to evolve highly complex compositional neural networks (CPPN) (Stanley et al 2009) having the regularity, modularity, and hierarchy properties for some problems.

3   USV System Architecture

In this work, we focus on the automated synthesis of an action selection policy for the USV to block the advancement of an intruder boat toward a particular target. For this task the USV is required to utilize reactive planning complemented by short-term forward planning to generate local action plans representing specific maneuvers. The developed policy synthesis approach is closely coupled to the underlying system architecture of the USV. The architecture consists of several modules that are responsible for different tasks, e.g. sensing, localization, navigation, planning, behavior control, communication, human interaction, or monitoring (Corfield and Young 2006; Finn and Scheding 2010).

The high simulation speed of the USV's dynamics model is critical for policy synthesis and therefore we use its simplified version with 3 degrees of freedom. This simplified model has been adapted from the full-blown 6 degrees of freedom USV dynamics model as described in (Thakur and Gupta 2010). The full-blown model considers disturbances from the surroundings and is used for high-fidelity physics-based real-time simulations inside the virtual environment.

3.1   USV Virtual Sensor Models

The planning system of the USV needs to only process relevant key sensory information abstracted from the raw sensor data. The sensory information is thus represented as a vector containing only the features required for a successful fulfillment of the mission task. The values of the relevant features are computed using the data from the virtual sensors (LaValle 2009) that
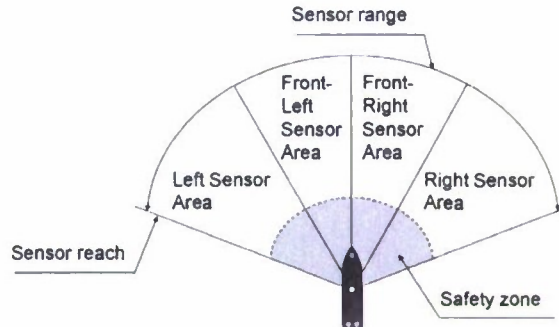
Figure 3: USV sensor model

provide intermediate abstraction of the raw sensor data. Some of the features can also have one or more parameters using which their final values are computed.

The planning system uses virtual visibility, relational, and velocity sensors. The virtual visibility sensor is a detection sensor with cone-shaped detection regions (see Fig. 3). The dimension of the overall sensor area is defined by its reach and range parameters. Each region returns a boolean value expressing the presence of other objects and a normalized distance to the closest object. The relational virtual sensor provides relevant information on how other objects are situated with respect to the USV or to each other. It provides boolean values for computation of the values of the relational features. The velocity virtual sensor returns the velocities of other objects inside the environment.

3.2 Planning Architecture

3.2.1 Policy Representation

The sequential decisions made by the vehicle depends on its action selection policy π. The policy π is defined in the context of the semi-Markov decision process (SMDP) (Ibe 2009) and consists of one default policy π_d and a set of specialized local policies π_l that map macro-actions to a specific set of overlapping regions $\Pi = \{S_1, ..., S_n\}$ of the state space S. The SMDP models temporally extended actions and is defined as a tuple $\langle S, A, T, R, F \rangle$, where

8

S is a finite set of states, A is a finite set of actions, $T : S \times A \times S \to [0,1]$ is a probabilistic transition function, where $T(s, a, \cdot)$ is a probability distribution for any $s \in S$ and $a \in A$, $R : S \times A \to \Re$ is a reward function that computes vehicle's immediate reward for taking action a in s, and F is a function giving probability of transition times for each state-action pair. The transition distribution is defined as $F(s', N|s, a)$ and specifies the expected number of steps N the action a takes starting in state s and terminating at state s'.

Hence, the policy π allows the USV to make a decision from a set of allowable actions based on observations of the world state. The default policy π_d is represented as a high-level controller (see Fig. 4). This controller is represented as a decision tree consisting of high-level parameterized commands NC, conditional variables CV, standard boolean values and operators BVO, program blocks PB, and system commands SC (see Tab. 1). The local policies are represented as action plans. The action plans are sequences of parametrized high-level action commands NC and program blocks PB. The action plans are internally represented as trees so that the same synthesis mechanism used to generate the default controller can also be used to generate the plans. The leaves of the decision tree can be conditional variables or action commands. The inner nodes can be conditionals, action commands, or system commands. Each action command corresponds to a particular high-level controller, which is a parameterized composition of simple behaviors according to the behavior-based control architecture (Brooks 1991). The section 3.2.3 describes this in detail.

The conditional variables, action, and system commands are parameterized. The parameters of an action command define its underlying property. The positional commands (e.g. *go-intruder-front*) are defined using 5 parameters. The first two parameters represent the USV's relative goal position (in polar coordinates) around the intruder. This effectively allows the vehicle to cover all feasible positions, as defined by its policy in a certain area around the intruder. The next two parameters represent a cone-shaped blocking area around the relative goal position. Once the vehicle gets inside the blocking area, it starts slowing down to limit the intruder's movement. The last parameter represents the length of the command execution. The turning action commands have two parameters that represent the turning rate and the length of the command execution. The translation velocity is explicitly controlled by the velocity commands. The *usv-sensor* system command effectively changes the parameters of the USV's sensors allowing it to explicitly control the risk level of the obstacle avoidance behavior. Each parameter of

| NC | go-intruder-front (front-left, left, front-right, right) |
	turn-left (right)
	go-straight
CV	intruder-on-the-left (right, front, at-the-back)
	intruder-has-target-on-the-left (right)
	usv-has-target-on-the-left (right)
	usv-intruder-distance-le-than
	usv-closer-to-target-than-intruder
	usv-facing-intruder
	usv-left (right, front-left, front-right)
BVO	if, true, false, and, or, not
PB	seq2, seq3
SC	usv-sensor, usv-velocity, usv-match-intruder-velocity

Table 1: Policy primitives

the command is propagated to the underlying primitive behaviors of each corresponding high-level controller.

The input into the policy is data from the virtual sensors. The values of all conditional variables are computed using this data. So for example, the boolean value of the *intruder-on-the-left* variable is directly provided by the virtual relation sensor while the data for computation of the *intruder-velocity-le-than* parameterized variable is provided by the virtual velocity sensor.

3.2.2 Policy Execution

During the mission, the USV periodically senses its surroundings and classifies its current state with respect to the intruder and the target. The classification mechanism of the policy executor decides whether the current state is close enough to one of the states for which a corresponding action plan exists. If such a plan exists, the policy executor directly executes the plan, otherwise it executes the default controller to generate a new sequence of actions. The decision whether to execute a specific plan depends on the activation distance parameter δ. This parameter defines the minimal distance that has to be achieved between the current USV's state and any state in the predefined set to activate a specific plan. The state space $S = S_1 \cup S_2$ is thus

10
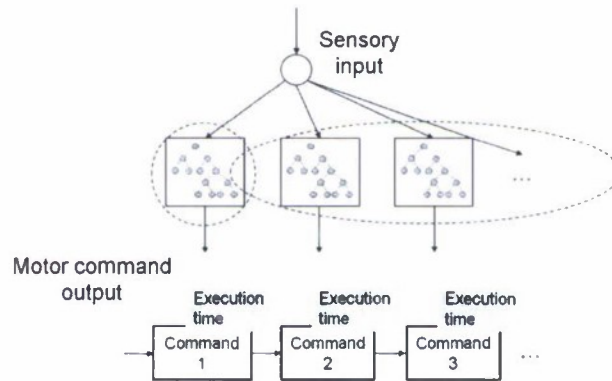
(a) Overall policy that consists of one default controller and a set of action plans. The product of the execution is a sequence of action commands (bottom)
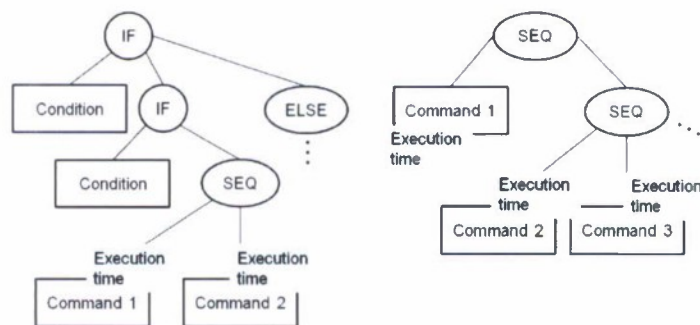
(b) Example of a default controller (left) and an action plan (right)
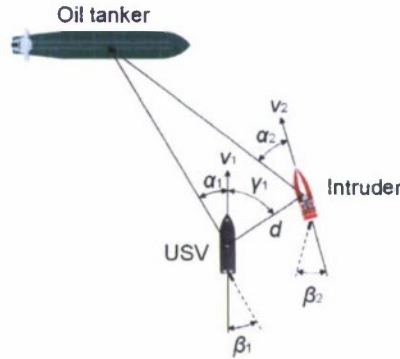
Figure 4: Policy representation

11

Figure 5: USV's state in respect to the intruder and target

divided into two regions where in the first region S_1 the USV generates and executes plans using the default controller, whereas in the other region S_2 the USV directly executes previously generated or manually defined plans. The action plan thus represents a certain maneuver the USV executes in a certain situation to increase its performance. The distance between normalized states is computed using the standard Euclidean distance metric.

The full state of the USV (see Fig. 5) is represented by a 8-dimensional vector of state variables $s = [\alpha_1, \alpha_2, \beta_1, \beta_2, v_1, v_2, \gamma, d]$ that encodes attributes of the environment for the task. The angle α_1 represents an angle between the USV's heading and the direction to the target, α_2 is an angle between the intruder's heading and the direction to the target, β_1 is the USV's steering angle, β_2 is the intruder's steering angle, v_1 is the USV's translation velocity, v_2 is the intruder's translation velocity, γ is an angle between the USV's heading and the direction to the intruder, and d is the distance between the USV and the intruder.

3.2.3 Hierarchical Control Architecture

By acquiring and processing sensor data in short-term cycles, and planning, the planning system determines an action command to be executed through the behavior-based control system to direct the vehicle inside the environment. The policy executor of the planning system takes as inputs sensor data, mission parameters, USV meta model, and the policy. It decides which

component of the policy to execute to generate a plan based on the current vehicle's state. The plan consists of a number of action commands, each being executed for a certain amount of time. The ultimate outputs of an activated command are way points that are directly translated by a low-level controller into motor commands for device drivers of a particular actuator.

The control architecture is hierarchical and follows the behavior-based paradigm (Brooks 1991). It consists of planning, executive, and reactive layers (see Fig. 6). The planning layer is responsible for interpreting stored policy and generating action plans. The commands are stored in a queue to be further dispatched for execution by the dispatcher in the executive layer. The commands are usually planned for short-term execution, such as planning of strategic maneuvers. The vehicle thus does not act purely reactively to its surroundings unless an exception occurs.

Each command corresponds to a high-level controller, which is a parameterized composition of simple behaviors organized into layers according to the behavior-based control architecture (Brooks 1991). The executive layer is responsible for processing the commands in the queue and invoking the corresponding high-level controllers in a series for planned periods of time. The length of the execution is defined as a parameter of the command. The executive layer is also responsible for monitoring execution of the controllers and handling exceptions. An exception occurs if the current state of the vehicle substantially deviates from the predicted trajectory defined by the plan. The policy executor remains inactive until all the commands from the queue are processed in which case the dispatcher requests new commands from the executor and the control process continues.

The reactive layer implements the behavior-based subsumption architecture (Brooks 1991). This architecture decomposes a complicated high-level controller into a set of simple behaviors (steer left / right, go straight, arrive) organized into layers. These primitive behaviors are finite state machines acting in response to sensor inputs and producing actuator action outputs. Multiple behaviors can be activated simultaneously producing different conflicting motor commands. This means that a certain amount of coordination is needed. Due to its robustness, we have chosen a priority-based arbitration mechanism, picking the actuator action output of the behavior with the highest priority as the overall action output of the currently activated high-level controller. This closely follows the behavior-competitive paradigm that imposes that only one behavior can have control over the robot's actuators while each of them can access all sensors. In this paradigm, the behavior in

13

the highest layer has the highest priority (for example obstacle avoidance) while the behavior in the lowest layer represents the most abstract functionality. In the architecture, each high-level controller specifies a fixed priority ordering of behaviors as defined by (Brooks 1986).

The primitive behaviors are of a great importance as they are able to quickly produce an action in a highly dynamic environment where fast response is crucial. A behavior is a simple unit that produces an output of a pre-defined type, in our case a two-dimensional vector containing desired translation velocity and steering angle. Conditions for activating behaviors are preprogrammed. The architecture defines the following primitive behaviors: obstacle avoidance, go to location, go straight, turn left, and turn right.

By default, the behaviors always choose such translation and steering velocities that maximize the USV's performance. So for example, *go-straight* behavior uses maximum translation velocity to get to the requested position in the shortest amount of time. The policy can override this by calling *usv-velocity* system command. This command switches the vehicle to its controlled velocity mode in which the USV's translation velocity is controlled by the higher-level policy.

3.2.4 Obstacle Avoidance

The behavior of many standard avoidance methods is driven by its carefully tuned parameters. These parameters control the behavior of the vehicle, particularly how much steering should be applied when a nearby obstacle is positioned at a certain distance and angle, and how fast the vehicle should be moving in that situation. Hence, for our mission task, the resulting behavior can be quite different with different parameters essentially controlling the vehicle's safe distance from the adversary and blocking efficiency at the same time. Insufficient avoidance steering can lead to collisions. On the other hand, too much steering will veer the vehicle away from the adversary leading to ineffective blocking.

The obstacle avoidance behavior is a necessary part of all high-level controllers. It uses the virtual visibility sensor (see Fig. 3) in order to identify the location of detectable obstacles. It directly produces desired translation velocity and steering angle to safely steer the vehicle away from the closest identified obstacles. The desired steering angle increases with the proximity to the obstacles while the translation velocity decreases. We have implemented a collision avoidance method that uses high-level sensory information, e.g.
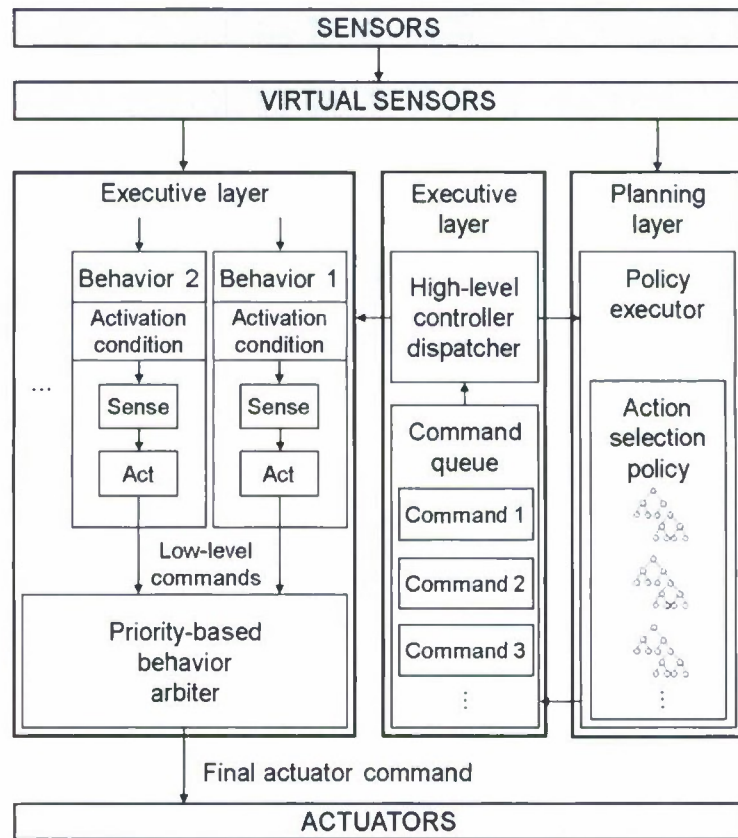
14

Figure 6: Hierarchical behavior-based control architecture

positions, orientations, and dimensions of obstacles, to directly decide which steering angle and translation velocity to request. The avoidance mechanism uses a fixed set of control parameters. However, the behavior can be conveniently controlled by modification of the underlying parameters of the visibility sensor. This way, the USV can get closer to obstacles than it would be otherwise possible and thus effectively define a balance between safe and aggressive maneuvering. The command *usv-sensor* of the policy modifies the reach and range parameters of the virtual visibility sensor cones. Moreover, as far as computational requirements are taken into account, the obstacle avoidance mechanism should be very fast so that it does not consume much of the simulation time.

4   Simulation Environment

4.1   New Design of the Simulation Environment

The Energetics Technology Center (ETC) is in the process of developing a USV simulation environment in support of several tasks assigned to the USV project. This software currently consists of four main components with. The four components are Virtual Environment-based (VE) Simulator, Controller Model, Mission Planner and the Sensor Model. The four modules are pictured in Fig. 7. The four modules may run within a single application on a single machine or as four separate processes, possibly on remote machines. The modules pass data using sockets along with send and receive queues. The queues are managed by separate threads as depicted by "T" in the diagram. The transfer rates are configurable. The numbers in the diagram were arbitrarily chosen. A centralized memory manager is used only when the modules run within a single application. One of the motivations for this design is to allow Hardware-in-the-loop (HIL) simulations to be done in the future.

In the older version of the system, the Mission Planner module was completely responsible for calculating the positions and orientations of the boats using an internal dynamics model. This was a simple car dynamics model. The Mission Planner then passed the positions and orientations on to the VE Simulator module, which then applied some simple physics calculations in order to modify the pose fed in by the Mission Planner and rendered the scene. The Mission Planner and VE Simulator used to be embedded in one
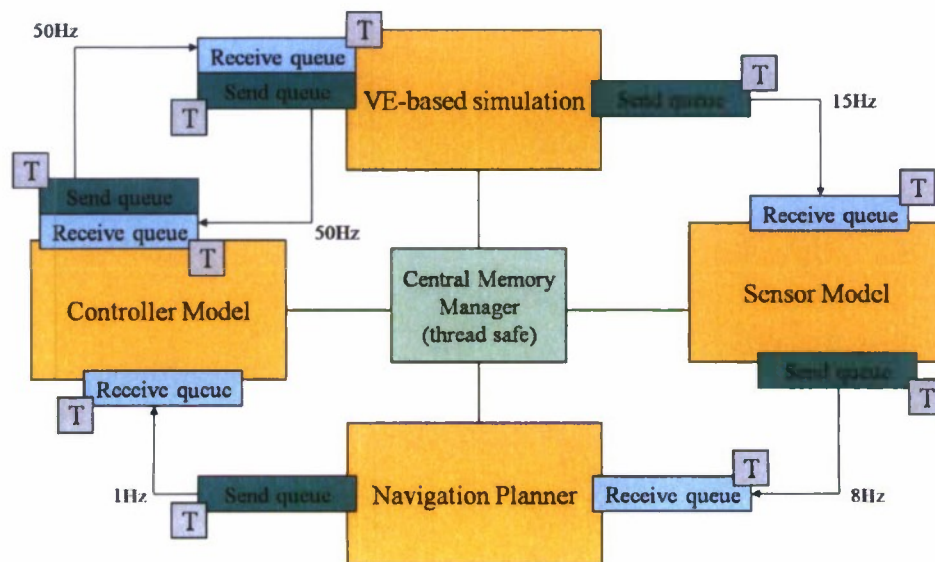
Figure 7: Four main modules of the USV simulation environment

application and communicated with each other using function calls instead of network sockets. Consequently, the monolithic application containing both software components quickly became challenging to manage and upgrade.

In the latest, redesigned, version of the system the Controller Model takes inputs for each boat from the Mission Planner module in the form of:

- object id

- desired speed

- desired acceleration

- desired angular speed

- *either* desired heading *or* waypoint (longitude, latitude)

The Controller then uses a built-in PID controller to calculate forces which need to be applied on the boat in order to meet the desired results. The forces – torque and drive force – are then sent to the virtual environment

17

based simulator which is responsible for using physics models to determine actual motion. The torque and thrust which the Controller calculates are roughly equivalent to forces the boat may experience from the propeller and rudder. There is a feedback loop from the VE Simulator to the Controller Model, supplying current pose, velocity, acceleration and other information.

Upon receiving the drive force and the torque from the Controller Model, the VE Simulator first calculates the motion of the boat based on just these two forces. Then, it has two options for calculating motion resulting from wave action. It can use the built-in high fidelity physics engine or it may use a simple high performance method of producing bobbing motion. After the motion has been calculated and rendered, the VE Simulator passes off the positions and orientations of all objects to the Sensor Model.

The Sensor Model is tasked with feeding sensor data into the Mission Planner, completing the loop. It receives ground truth from the VE Simulator. The Sensor Model loads the collision mesh of the object, which also serves as a point outline, from a shared configuration file. The Sensor Model then applies a transformation, based on positional data it received from the VE Simulator, on each of the outline points. The transformation takes into account the perspective of the USV or intruder in order to rotate and translate the points representing terrain and boat obstacles. Those points are then shipped to the Mission Planner. The intent is for the Mission Planner to make collision avoidance decisions based on terrain and obstacle polygons arriving over the network. One of the tasks of the Sensor Model is to first add some error into the data. Currently, data is not altered, however. Future work will address the issue of adding noise and error to the data.

The communication between the four modules is carried out over the network. The modules use connectionless datagram sockets for fast communication. Data is sent over the network in binary format to reduce the payload size. Four classes encapsulate the messages sent from each module. The modules can access the data inside the messages simply by calling accessor functions. Whereas the real controller will likely have access to a GPS receiver, it was important for the communication interface to support transmission of realistic longitude and latitude. Representing position data using longitude and latitude is a good way to standardize exchange of positions in general because longitude and latitude are well known and well defined means of specifying global locations. The size of the message classes had to be expanded so as to accommodate the new size of position data. Both latitude and longitude had to be transmitted as doubles instead of floats in

order to allow conversion to meters. Due to the fact that each longitude or latitude degree represents a distance that is far larger than a meter, these two measurements had to be encoded in a data type with very high precision (e.g. support for high number of decimal digits). Internally the modules use meters. The conversion is made using a set of convenient static functions.

In addition to the separation of the four main components of the system and the redesign of the communication mechanism they use, several modules have also been internally upgraded. The virtual environment simulator module no longer has hardcoded terrain models. Rather, the configuration file and the class which loads it have been expanded to allow a comma separated list of terrain files. The obstacle positions are also no longer hardcoded. Obstacle data is now stored in separate files, one file per obstacle. The main configuration file has a comma separated list of obstacle configuration files. The class which loads the configuration files has been made available to all four modules. This allows all modules to initialize their environment with the same shared data.

4.2   Development of the Infrastructure for Physical Testing of Generated Policy

Significant coding effort has been spent on the second version of the image processing application. This application will be used to keep track of model boats. The environment in which these small boats are meant to operate is a 50 foot wide tank within the Neutral Buoyancy Research Facility (NBRF) at University of Maryland. Although the first version was fully operational, it was very user unfriendly and difficult to configure. This configuration difficulty is what led to several human errors in the past, resulting in a lot of lost time diagnosing and then reconfiguring the system at the NBRF. The second version of the system was designed to fix these problems by allowing easy reconfiguration of the system with a lot of automatic computations, taking the potential for user error out of many configuration tasks. In the month of July, all the coding work was related to user interfaces of the application (e.g. windows, dialogs, context menus, etc.) and the communication between them. During most of the month of August, work on this project at ETC dealt with wrapping up the implementation and testing of the second version of the image processing software used to keep track of model boats. The objective of the second version of the software was mainly to streamline

19

and optimize the tracking and calibration process. Work on this image processing application falls under Task #14 – validate and assess the quality of computer generated software by conducting tests on USSV-HTF.

The first version of the software was divided into two separate components. The tracking component was built at ETC while the calibration component was created at UMD. Much of the input needed for the calibration component had to be generated by hand in the first version of the system. This opened up many opportunities for human error and slowed down the calibration process. Output produced by the calibration component, in the form of a binary file, was used as input in the tracking component. All tracking attributes and settings from the number of targets and their colors to world coordinate adjustments were static – tracking application had to be restarted, after changes to a configuration file were made, for new changes to take effect.

The second version of the software solved these problems by (1) fully integrating the calibration module into the tracking software, (2) calculating much of the calibration data automatically, and (3) providing a mechanism for changing the tracking attributes dynamically. The latest version of the software has streamlined the tracking and calibration process in two tangible ways:

- Once the physical setup is in place, software calibration can now be performed in about 10 minutes instead of 25 minutes.

- As a result of heavy use of multi-threading, tracking speed has been raised from 10 fps to 14 fps.

4.3 Modeling of Shorelines

The last week of August was used to work on Task #8 – incorporate models of shorelines in the simulation environment. ETC was able to take advantage of some talented new personnel within the organization to create new models of terrain – a task previously performed by interns. The models of terrain were created using a software tool called LightWave and then converted into a file format used by the USV simulation environment. Although there is a reliable process for translating CAD models of terrain created with LightWave and Blender into the file format used in the simulation environment, the process still requires some hand tuning and often takes a good amount of time. In

addition to representing a shoreline with a CAD model inside the simulation environment, appropriate collision handling had to be implemented. Collision detection has been enabled between boats and terrain, in addition to collision detection between just boats. The same simple collision physics which was used for boat-to-boat interactions in phase 1 of this project is now used for boat-to-terrain interactions.

4.4 Additional Visualization Features of the USV Simulation Environment

Improvement of the rendering quality of the ocean is something that has been gradually carried out at ETC, often during a brief lull in activities. Previously, the ocean geometry had a single water texture applied to it. The texture was static. It did not shift based on the motion of the waves. Various lighting, texture and color calculations for rendering the ocean were also performed mainly on the CPU using the Ogre rendering engine. In July, new features were added which shifted many calculations to the GPU using OpenGL Shading Language (GLSL). Currently, most of the lighting and color calculations are performed using OpenGL vertex and fragment shaders. This enables a much more fine grained level of control over how the ocean is rendered. These techniques will allow many other improvements in the future. The most immediate improvements are multi-texturing and dynamic shifting of textures. The rendered appearance has been improved with the addition of a second sky texture which is blended with the water texture. Texture coordinates for both textures are now continuously recalculated every frame update to shift based on the motion of the waves, making the water look more "alive". One added side effect of integrating GLSL into the USV simulation environment is that much of the ocean physics calculations may be performed on the GPU (implemented in GLSL) in the future. This should allow us to significantly boost the speed of high fidelity physics simulations at some point down the road. Some preliminary planning has already started between the UMD team and ETC team. Fig. 1 presents a scene from the simulation environment with improved graphics.
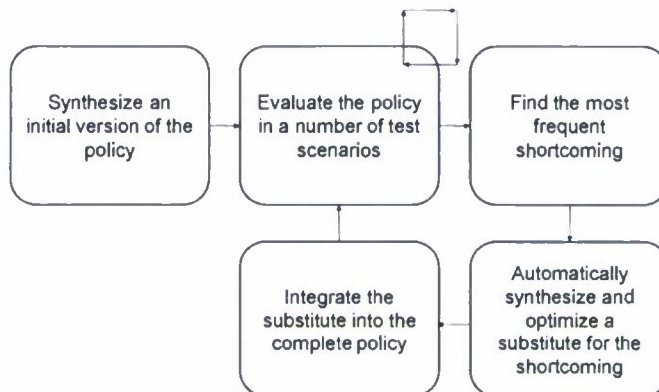
21

Figure 8: Policy synthesis overview

5 Policy Synthesis

In this section, we describe our approach for policy synthesis that can be viewed as a completely automated process during which an initial version of the policy is automatically synthesized and then gradually refined by detecting and fixing its shortcomings (see Fig. 8). In this iterative process, the states in which the vehicle has a high probability of failing its task are identified and new specific action sequences are generated for the vehicle to avoid these failure states in future planning. Plans are incrementally synthesized for a sequence of representative states of exception and their immediate neighborhoods in which the vehicle has the highest potential of avoiding the largest amount of the detected failure states. In this way, the new plans are incrementally associated with specific isolated groups of nearby states to improve the efficiency of the default policy. The synthesis process takes the advantage of the fact that a large proportion of the state space is not encountered during the actual policy execution so that the possible state explosion can be avoided. A diagram describing a detailed portion of the the overall synthesis process is depicted in Fig. 9. It particularly shows the evaluation – action plan generation section of the whole cycle (see Fig. 8) as discussed next.

The actual algorithm for computing the policy takes input as a SMDP model and computes a stationary policy $\pi : S \rightarrow A$ that attempts to maxi-

mize expected accumulated reward. The main steps of the algorithm are as follows:

1. Employ GP to evolve an initial version of the policy π_d.

2. Evaluate π_d inside the simulator in m distinct evaluation runs. This evaluation returns a set of states of failure $S_f = \{sf_1, ..., sf_n\}, n \leq m$ in which the vehicle fails its mission task.

3. Given S_f, compute a corresponding set of states of exception $S_e = \{se_1, ..., se_n\}$ in which proximity (given by the activation distance parameter δ) the vehicle has the potential to avoid the states of failure if it executes appropriate corresponding action plans.

4. Compute a representative state of exception $se_{rep} \in S_e$, as described in detail in Section 5.1, in which the vehicle has the potential to avoid the largest amount of the detected failures. The vehicle thus can execute a specific action plan in se_{rep} and its neighboring states to decrease the probability of occurrence of the corresponding states of failure. In this way, the focus is always restricted to the most critical states of failure first while leaving the rest for possible later processing.

5. Employ GP to synthesize a new specific action plan for the representative state of exception se_{rep}. To prevent over-specialization of the new plan, evaluate its performance using all states of exception found within se_{rep} activation distance δ during the overall policy evaluation in step 2.

6. Optimize the new plan and integrate it into the whole policy together with its corresponding se_{rep}. If the termination condition is not satisfied, continue to step 2. The distance between the normalized states is computed using the Euclidean distance metric.

5.1 Extraction of a representative state of exception

The overall algorithm identifies states of failure S_f in which the vehicle has a high probability of failing its task due to the execution of inappropriate actions. The algorithm for extraction of the representative state of exception se_{rep} (see Alg. 1) always extracts a state in which the vehicle can execute an

appropriate action plan to avoid the largest portion of the detected states of failure.

The algorithm iterates over all states of exception $se \in S_e$ and for each of them finds its neighboring states $se_{Neighbors} = \{se_{Neighbor} | \; ||se, se_{Neighbor}|| \leq \delta\}$ within the activation distance δ. Then, for se and each neighboring state from the set $se_{Neighbors}$, a corresponding state of failure sf is found and all its neighbors $sf_{Neighbors} = \{sf_{Neighbor} | \; ||sf, sf_{Neighbor}|| \leq \delta\}$ within the distance δ. The algorithm terminates by returning se that is associated with the largest amount of corresponding states of failure.

Algorithm 1 Extraction of a representation state of exception se_{rep}

Require: States of exception S_e, states of failure S_f

  $se_{rep} \leftarrow NIL$
  $size_{max} \leftarrow 0$
  for all $se \in S_e$ **do**
    $se_{Neighbors} \leftarrow$ FINDNEIGHBORS(se, δ)
    $S_f \leftarrow \emptyset$
    for all $se_{Neighbor} \in se \cup se_{Neighbors}$ **do**
      $sf \leftarrow$ FINDCORRESPONDINGFAILURE($se_{Neighbor}$)
      $sf_{Neighbors} \leftarrow$ FINDNEIGHBORS(sf, δ)
      $S_f = S_f \cup sf_{Neighbors}$
    end for
    if $|S_f| > size_{max}$ **then**
      $size_{max} \leftarrow |S_f|$
      $se_{rep} \leftarrow se$
    end if
  end for
  return se_{rep}

In the context of our mission task, a state of failure defines a situation in which the USV has a high probability of losing a future chance to block the intruder. Its corresponding state of exception is found by reverting back in time for a predefined number of time steps τ to record a state from which a new specific action plan can be executed to prevent a possible future failure. This simple way of determining a state of exception can be further improved by developing a special task-related heuristics that precisely determines a state from which the vehicle will have the highest chance of successfully avoiding the largest amount of possible failure states.
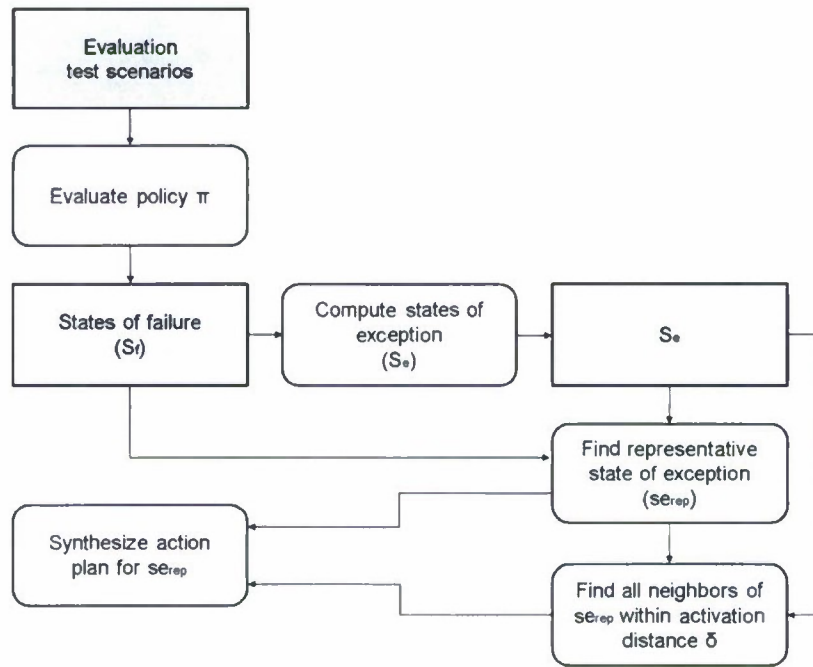
Figure 9: Extraction of a representative state of exception se_{rep}

5.2 Evolution

The evolution searches through the space of possible plan configurations to find the best possible action plan for a particular state. Both the default controller and specialized action plans as components of the policy are automatically generated using separate simulated evolutionary processes. The specific evolutionary method we used is the strongly-typed Genetic Programming imposing type constraints on the generated Lisp trees (Poli et al 2008). This is a robust stochastic optimization method that searches a large space of candidate program trees while looking for the one with the best performance (fitness value).

The evolutionary process starts by randomly generating an initial population of individuals represented as GP trees using the Ramped half-and-half method (Poli et al 2008). The initial values of parameters of all action commands and conditionals are either seeded or randomly generated. The default controller of the policy is generated using a human written template for which GP supplies basic blocking policy. The first portion of the template encodes a maneuver using which the vehicle effectively approaches the intruder at the beginning of the run as there is no need for it to be explicitly evolved.

The terminal T and function sets F consists of action commands, system commands, conditional variables, boolean values and operators, and program blocks as defined in section 3.2.1. The sets are defined as

$$T_{controller} = T_{plan} = NC \cup SC$$
$$F_{controller} = CV \cup BVO \cup BP; F_{plan} = PB$$

Within the population, each individual has a different structure responsible for different ways of how it responds to its environment. The individuals are evaluated in the context of the whole policy inside the simulator. The sensory-motor coupling of the individual influences the vehicle's behavior resulting in a specific fitness value that represents how well the USV blocks the intruder.

We favor individuals which can rapidly establish basic blocking capabilities and optimize them to push the intruder away from the target over the entire trial duration. To do so, the fitness F is defined as the sum of normalized squared distances of the USV from the target over all time steps. If a collision occurs, either caused by the USV or the intruder, the zero fitness value is assigned to the individual, and the selection pressure eliminates

the policy component with low-safety guarantee. The fitness function is as follows:

$$F = \frac{1}{T} \sum_{i=1}^{T} \left[\frac{d_i}{d_0} \right]^2 \tag{1}$$

where T is the total number of time steps, d_i is the distance of the intruder from the target at time step i, and d_0 is the initial distance of the intruder from the target in a particular test scenario. The total fitness value of the individual is maximized and computed as an average of the fitness values resulting from all scenarios.

The default controller is evaluated using a human-competitive intruder in 8 different scenarios. In each scenario, the intruder has a different initial orientation, and the USV always starts from an initial position closer to the target. The evaluation lasts for a maximum amount of time steps which equals to 300 seconds in real time. The maximum speed of the USV is set to be 10% higher than the speed of the intruder, other properties of the vehicles are the same. The action plan is evaluated using all states of exception found within the activation distance of its corresponding se_{rep}. The evaluation lasts for a maximum amount of time steps which equals to 10 seconds in real time.

The individuals in the initial population mostly exhibit random behavior. By selecting and further refining the individuals with high fitness, their quality gradually improves in subsequent generations. During this process, the individuals are randomly recombined, mutated, or directly propagated to the next generation. These operations are applied with the predefined probabilities (see Tab. 2). The following evolutionary operators are used:

1. Reproduction – copies one individual directly to the next generation without any modification. We use a strong elitism to propagate the best individual directly into the next generation. This makes sure that the best individual is not lost during the evolutionary process as a consequence of recombination.

2. Mutation – we use three types of mutation operators: structural mutation of a randomly selected sub-tree, preventing bloat (Poli et al 2008) by shrinking a randomly chosen sub-tree to a single node, and Gaussian mutation of chosen parameters.

3. Crossover – randomly selects sub-trees from two input trees and swaps them.

Population size / number of generations	500 / 100 (controller) 50 / 30 (plan)
Tournament size	2
Elite set size	1
Min. and max. initial GP tree depth	3 and 6 (controller) 2 and 4 (plan)
Maximum GP tree depth	50 (controller) 10 (plan)
Crossover prob.	0.84
Structure mutation prob.	0.05
Shrink structure mutation prob.	0.01
Mutation prob. of parameters of action commands	0.5
Crossover probability	0.84

Table 2: GP parameters

During the policy synthesis, the USV learns the balance between a safe and dangerous maneuvering by mutating the reach and range parameters of its virtual visibility sensor. The policy is thus co-evolved with the sensor parameters of the vehicle to control the obstacle avoidance mechanism.

The optimization of the generated default controller removes all branches of the code that have not been executed during evaluation scenarios. Moreover, each generated action plan is truncated to contain only the action commands that do not exceed the maximum execution time of the plan.

A detailed description of the functionality of all the operators used can be found in (Koza 2003). The control parameters of the evolutionary process used for evolution of the default controller and plans are summarized in Tab. 2.

6 Experiments

6.1 General Setup

We tested the developed approach in the context of a combat mission task during which the USV protects a valuable target against an intruder boat. In this task, the intruder boat has to reach the target as quickly as possible while the USV has to block and delay the intruder for as long time as possible. We set up an experiment to compare the performance of the automatically generated USV's policy to the USV's policy coded by hand. We compare the performance in terms of pure time delay imposed by the USV on the intruder. To get a fair assessment of the USV performance, the time values being compared must be normalized by 40 seconds baseline. This baseline represents the amount of time needed to reach the target if the intruder is completely unobstructed. Any additional time above this baseline thus represents the effective delay time of the intruder when being blocked by the USV.

The policy of the USV is evaluated in 800 runs to account for the intruder's nondeterministic behavior. Each evaluation run lasts for a maximum amount of time steps which equals to 300 seconds in real time. The dimension of the scene is 800×800 m with the target positioned in the center. At the beginning of each run, the USV and the intruder are oriented toward each other with random deviation of 10 degrees and the USV is positioned on a straight line between the intruder and the target. The initial distance of the USV from the target is approximately 240 m while the intruder's initial distance is 360 m. The maximum time for the evaluation run is set to 5 minutes. The USV's maximum velocity is 10 m/s while the intruder's maximum velocity is 9 m/s.

6.2 Experimental Protocol

First, we manually implemented an initial version of the intruder's attacking policy and tested it against human players to evaluate its performance in the virtual environment. The policy was further improved in multiple iterations in the span of 6 weeks. Its overall size reached 485 lines of Lisp code. The outline of the policy functionality is described in the next section. We evaluated the performance of the policy by pitting human players against it playing as USVs. The human players achieved 90 seconds of pure time delay

29

imposed on the intruder in average. This shows that the intruder's attacking policy is quite sophisticated as it exhibits a deceptive behavior.

Second, we implemented the USV's blocking policy against the hand coded intruder. The policy was improved iteratively in the span of 3 weeks. Its overall size reached 500 lines of Lisp code.

Third, we used the mission planning system to automatically generate the USV's blocking policy using the hand coded intruder as the competitor. The activation distance parameter δ was set to 0.2 for all action plans. In the current version of the approach, a state of failure is determined to be the state in which the intruder is closer to the target than the USV. A state of exception is computed by going back in time for 150 time steps from a given state of failure.

Finally, we compared the performance of the automatically synthesized USV's policy to the hand coded one.

6.3   Intruder's Policy

The design of the intruder's attacking policy was a crucial step during the initial stages of the experimental setup. The level of aggressiveness of the intruder's attacking behavior is defined such that the intruder presents a high challenge for human players playing as USVs but at the same time executes relatively safe obstacle avoidance.

The evaluation scenario is nondeterministic, i.e. the intruder may execute different actions in the same state which may lead to different outcomes. The nondeterminism of the intruder's policy thus allows the intruder to repeatedly deceive the USV during the combat so that the USV is not able to find a motion pattern in intruder's behavior that can be easily exploited for blocking.

The nondeterminism of the intruder's behavior puts a great challenge on the USV policy synthesis since the same USV's policies may acquire different fitness values when evaluated in the same test scenario. Thus for the purpose of fair evaluation, we use one specific type of the intruder's policy for computing the fitness values of all USV individuals from one specific population. The behavior expressed by the intruder is influenced by a set of randomized action commands that are parts of its policy. The randomized commands take a random number as one of their inputs based on which they produce an action. The random number is generated by a random number generator that is initiated by explicitly provided random seed. The random seed thus

indirectly defines a particular type of the intruder's policy and is kept to be the same during evaluation of USV individuals within the same population.

The hand coded intruder's policy is represented as a single decision tree that contains standard action commands as well as their randomized versions. The intruder's policy can be divided into five main sections. Each of these sections handles a different group of situations that can arise during the combat. The first section handles situations in which the distance of the intruder from the target is larger than 130 m and the angle between its translation direction and the target is more than 80 degrees. In these situations, the intruder attempts to rapidly change its direction of movement toward the target by aggressively turning left or right depending on the position of the USV.

The second section handles situations in which the USV is very close to the intruder, positioned relatively to its front left, and the target is on the intruder's left side (see Fig. 10a left). In this case, the intruder has two options. Either it executes a random turn with probability 0.9 or it proceeds with a complete turn. In both cases, the intruder can slow down rapidly with probability 0.3 to further confuse the adversary. This section handles also similar type of situations when the USV is on the front right of the intruder and the target is on the right.

The third section is very similar to the second section with the exception that the USV is directly on the left or right side of the intruder (see Fig. 10a right). In these cases, the intruder deceives the USV by randomly slowing down to get an advantageous position, proceeding with a complete turn, or executing a partial turn. The probability of the complete turn is 0.1 and the probability of slowing down is 0.2.

The fourth section deals with the situations during which the intruder is positioned behind the USV inside the rear grey area as shown in Fig. 10b left. The larger distance of the intruder from the USV gives it opportunity to exploit the USV's tendency to over shoot a little in the process of blocking. In this case, if the USV has high velocity, the intruder slows down and turns toward the stern of the blocking USV, passing the USV from behind. Otherwise, the intruder randomly turns with probability 0.7 or it proceeds with a complete turn (see Fig. 10b right). Again, the intruder can rapidly slow down with probability 0.3.

Finally, if the intruder is not in a close proximity to the USV, it computes the best sequence of actions in order to get to the target as fast as possible.

The intruder's policy can modify the reach and range parameters of its

31

virtual visibility sensor to control the balance between a safe and aggressive obstacle avoidance mechanism. For example, if the intruder wants to make an aggressive turn in a close proximity to the USV it has to take risk by decreasing the reach of the sensor to be able to quickly proceed with the turn. In this case, the sensitivity of the obstacle avoidance behavior is reduced for a short period of time so that the intruder can easily pass the USV from behind. If the intruder always aimed to safely avoid the adversary, it would not get any chance to get to the target, especially if it competes against a human player.

6.4 Results and Discussion

The experimental run that generated a blocking policy with the highest performance is shown in Fig. 11. The horizontal axis of the graph shows different versions of the policy consisting of gradually increasing amount of action plans. The vertical axis shows the blocking performance in terms of the intruder's pure time delay for each version of the USV's policy. The best performance is reached by the version 82 of the policy and amounts to 47 seconds of pure delay in median. This can be compared to the pure time delay of 46 seconds in median imposed by the hand coded USV on the same intruder. This result thus shows that the best performance of the automatically generated USV's policy exceeds the blocking performance of the hand coded policy.

The automated generation of the policy took approximately 1 day to generate the default controller and approximately 5 days on the average to generate action plans for 81 automatically defined states of exception on a machine with configuration Intel(R) Core(TM)2 Quad CPU, 2.83 GHz. Its overall size reached 900 lines of code. From the set of 10 experimental runs, only 2 were able to find a policy with the similar performance to the best one. The remaining 8 runs prematurely stagnated due to over-specialization of some of the evolved action plans. Even a single defective action plan synthesized for one of the key situations can significantly influence the performance of the whole policy. This shows that the synthesis of a policy for the USV to block the intruder utilizing a nondeterministic attacking policy is a challenging task.

The results show that the performance of the first few versions of the policy is low as they contain only a few action plans describing specialized maneuvers for a small number of key situations. However, as the synthesis

32
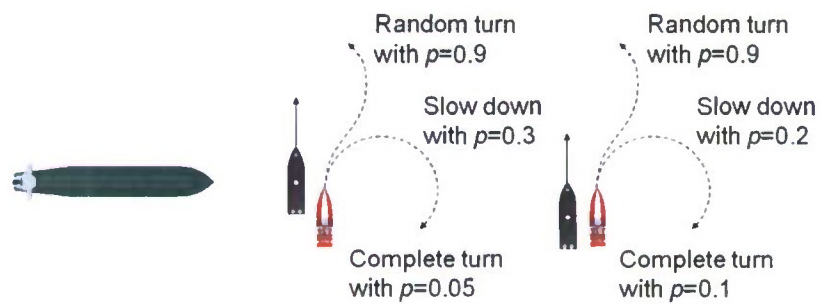
(a)
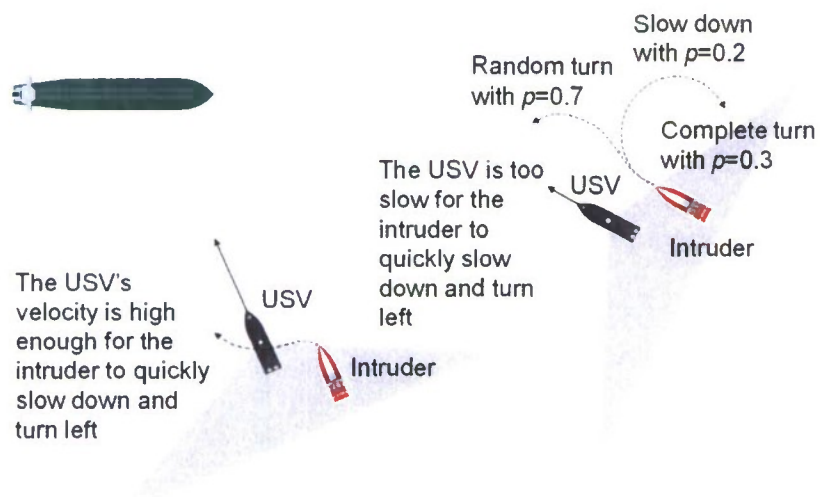
(b)

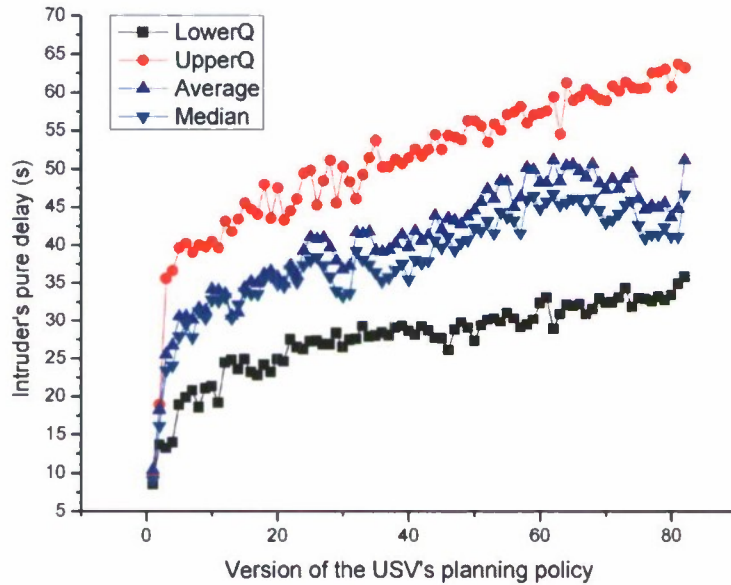Figure 10: Representative portions of intruder's policy

33

Figure 11: Evaluation of the USV's blocking performance. The performance is expressed as a pure time delay applied on the intruder. Each version of the USV's policy was evaluated in 800 runs.

process progresses, more and more action plans handling new situations are added and the overall performance gradually improves. This way the initial policy becomes sophisticated due to newly evolved action plans. The synthesis process continues until the version 82 of the policy after which the performance stagnates. This can be attributed to difficulty in solving new complex situations in which problems with the generalization of action plans arise.

Evolution of a single policy against an adversary exhibiting nondeterministic behavior thus generates a highly suboptimal solution. To further improve the performance of the policy, distinctive states are automatically isolated for which optimal short-term action plans are generated. As a result of that, the final policy demonstrates a clear performance advantage over the policy without the optimal substitutes.

An example of a run in which the USV reached 45 seconds of pure time
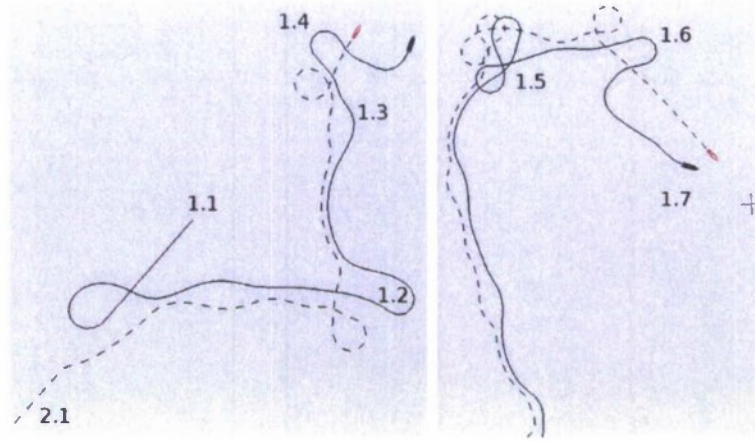
34

Figure 12: Example of a run in which the USV managed to block the intruder for 45 seconds. The start position of the USV is marked as 1.1 while the start position of the intruder is marked as 2.1.

delay imposed on the intruder is shown in Fig. 12. The USV starts at the location 1.1 while the intruder starts at the location 2.1. The first situation in which the USV executes a specific maneuver is marked as 1.2. In this situation, the USV steers sharply to the left in an attempt to intercept the intruder. The run continues until 1.3 where the USV attempts to deflect the intruder's heading by first carefully turning to the left and then aggressively blocking from the side. The intruder, however, instantly responds by executing a sharp left turn, which makes the USV to take another trial in intercepting him in the situation 1.4. Yet the USV overshoots in the process of blocking. The run continues for the next 23 seconds all the way up to the target. In the situation 1.5, the intruder executes a random sequence of two sharp turns to deceive the USV and thus to increase its chances for the attack. The USV, however, successfully follows and takes another attempt in intercepting the intruder but overshoots in 1.6 and the intruder finally reaches its goal 1.7.

7 Main Accomplishments

The main accomplishments for Year #1 include:

1. Developed a new general approach for automated action selection policy synthesis for unmanned vehicles operating in continuous state-action spaces.

2. Utilized the approach in Item #1 for automated synthesis of an action selection policy for USV to block the advancement of an intruder boat toward a valuable target. The intruder is human-competitive and exhibits a deceptive behavior so that the USV cannot exploit any regularity in its attacking behavior.

3. Developed a customized hierarchical behavior-based planning architecture utilizing reactive and forward planning for USVs.

4. Developed a simulation environment that integrates various components of the project into a complete simulation system and acts as a simulation platform for the synthesis module.

5. Performed performance comparison of the hand coded and automatically generated USV's blocking policy. The performance of the automatically generated policy exceeded the performance of the hand coded policy.

8 Future Plans

We do not anticipate any change to the plan presented in the proposal. During Year #2, we plan to work on Task #5, Task #6, Task #7, Task #8, and Task #10 described in the proposal.

9 Conclusions

We have presented a new approach for automated action selection policy synthesis for unmanned vehicles operating in adverse environments. The idea behind this approach is to generate an initial version of the policy first and then further improve its performance by evolving additional components that

increase the performance of the policy in certain states. These components are represented as short-term action plans that can reliably handle specific local failure situations that can arise during the mission.

Our particular focus was on the synthesis of a symbolic policy for an autonomous USV operating in a continuous state-action space with a deceptive adversary. We have developed a mission planning system to automatically generate a policy for USV to block the advancement of an intruder boat toward a valuable target. The intruder is human competitive and exhibits a deceptive behavior so that the USV cannot exploit any regularity in its attacking action rules for blocking. The USV's policy consists of a controller and multiple specialized action plans describing specific maneuvers that allow the USV to rapidly execute sequences of high-level commands in specific situations.

In our experiments, we compared the performance of the hand coded USV's blocking policy to the performance of the policy that was automatically generated. The results show that the performance of the automatically generated policy (47 seconds of pure delay in median) exceeds the performance of the hand coded policy (46 seconds in median). Both of the USV's policies were evaluated against the same set of manually implemented human competitive intruders. Hence, the approach described in this report clearly demonstrates the viability of our approach for automatically synthesizing symbolic policy for autonomous unmanned vehicles operating in highly dynamic environments.

References

Bacardit J, Bernadó-Mansilla E, Butz M (2008) Learning Classifier Systems: Looking Back and Glimpsing Ahead. Learning Classifier Systems pp 1–21

Baker C, Ferguson D, Dolan J (2008) Robust Mission Execution for Autonomous Urban Driving. Intelligent Autonomous Systems 10: IAS-10 p 155

Barate R, Manzanera A (2007) Automatic design of vision-based obstacle avoidance controllers using genetic programming. In: Proceedings of the Evolution artificielle, 8th international conference on Artificial evolution, Springer-Verlag, pp 25–36

Barlow G, Oh C (2008) Evolved Navigation Control for Unmanned Aerial Vehicles. Frontiers in Evolutionary Robotics pp 596–621

Board N (2005) Autonomous vehicles in support of naval operations. National Research Council, Washington DC

Brooks R (1986) A robust layered control system for a mobile robot. IEEE journal of robotics and automation 2(1):14–23

Brooks R (1991) Intelligence without representation. Artificial intelligence 47(1-3):139–159

Buason G, Bergfeldt N, Ziemke T (2005) Brains, bodies, and beyond: Competitive co-evolution of robot controllers, morphologies and environments. Genetic Programming and Evolvable Machines 6(1):25–51

Buhmann M (2001) Radial basis functions. Acta numerica 9:1–38

Corfield S, Young J (2006) Unmanned surface vehicles–game changing technology for naval operations. Advances in unmanned marine vehicles pp 311–328

Dain R (1998) Developing mobile robot wall-following algorithms using genetic programming. Applied Intelligence 8(1):33–41

Diederich J, Tickle A, Geva S (2010) Quo Vadis? Reliable and Practical Rule Extraction from Neural Networks. Advances in Machine Learning I pp 479–490

Doherty D, ORiordan C (2006) Evolving agent-based team tactics for combative computer games. In: AICS 2006 17th Irish Artificial Intelligence and Cognitive Science Conference

Dupuis J, Parizeau M (2006) Evolving a Vision-Based Line-Following Robot Controller. In: IEEE Proceedings, Citeseer

Finn A, Scheding S (2010) Developments and Challenges for Autonomous Unmanned Vehicles: A Compendium. Springer

Floreano D, Mattiussi C (2008) Bio-inspired artificial intelligence: theories, methods, and technologies

Gajda P, Krawiec K (2008) Evolving a vision-driven robot controller for real-world indoor navigation. In: Proceedings of the 2008 conference on Applications of evolutionary computing, Springer-Verlag, pp 184–193

Goldberg D (1989) Genetic Algorithms in Search and Optimization

Haynes T, Sen S (1996) Evolving behavioral strategies in predators and prey. Adaption and Learning in Multi-Agent Systems pp 113–126

van Hoorn N, Togelius J, Schmidhuber J (2009) Hierarchical controller learning in a first-person shooter. In: IEEE Symposium on Computational Intelligence and Games (CIG 2009), pp 294–301

Ibe O (2009) Markov processes for stochastic modeling. Academic Press

Jaskowski W, Krawiec K, Wieloch B (2008) Winning ant wars: Evolving a human-competitive game strategy using fitnessless selection. In: Genetic programming: 11th European Conference, EuroGP 2008, Naples, Italy, March 26-28, 2008: proceedings, Springer-Verlag New York Inc, p 13

Kohl N, Miikkulainen R (2008) Evolving neural networks for fractured domains. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation, ACM, pp 1405–1412

Koza J (2003) Genetic programming IV: Routine human-competitive machine intelligence. Kluwer Academic Pub

Koza J, Rice J (1992) Automatic programming of robots using genetic programming. In: Proceedings of the national conference on artificial intelligence, Citeseer, pp 194–194

Lanzi P (2008) Learning classifier systems: then and now. Evolutionary Intelligence 1(1):63–82

LaValle S (2009) Filtering and planning in information spaces. IROS tutorial notes

Lipson H, Antonsson E, Koza J, Bentley P, Michod R, Alber R, Rudolph S, Andronache V, Scheutz M, Arzi-Gonczarowski Z, et al (2003) Computational synthesis: from basic building blocks to high level functionality. In: Proc. Assoc. Adv. Artif. Intell. Symp, pp 24–31

Nehmzow U (2002) Physically embedded genetic algorithm learning in multi-robot scenarios: The PEGA algorithm. In: 2nd International Workshop on Epigenetic Robotics: Modelling Cognitive Development in Robotic Systems, Citeseer

Poli R, Langdon W, McPhee N (2008) A field guide to genetic programming. Lulu Enterprises Uk Ltd

Richards M, Whitley D, Beveridge J, Mytkowicz T, Nguyen D, Rome D (2005) Evolving cooperative strategies for UAV teams. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM, p 1728

Ryan J, et al (2009) Learning classifier systems: a complete introduction, review, and roadmap. Journal of Artificial Evolution and Applications 2009

Schrum J, Miikkulainen R (2009) Evolving multi-modal behavior in NPCs. In: Proceedings of the 5th international conference on Computational Intelligence and Games, IEEE Press, pp 325–332

Schwartz M, Svec P, Thakur A, Gupta S (2009) Evaluation of Automatically Generated Reactive Planning Logic for Unmanned Surface Vehicles. In: Performance Metrics for Intelligent Systems Workshop (PERMIS'09)

Shichel Y, Ziserman E, Sipper M (2005) GP-robocode: Using genetic programming to evolve robocode players. Genetic Programming pp 143–154

Sipper M, Azaria Y, Hauptman A, Shichel Y (2007) Designing an evolutionary strategizing machine for game playing and beyond. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37(4):583–593

Stanley K, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2):99–127

Stanley K, D'Ambrosio D, Gauci J (2009) A hypercube-based encoding for evolving large-scale neural networks. Artificial Life 15(2):185–212

Sutton R, Barto A (1998) Reinforcement learning: An introduction. The MIT press

Svec P, Gupta S (2010) Automated planning logic synthesis for autonomous unmanned vehicles in competitive environments with deceptive adversaries. In: New Horizons in Evolutionary Robotics: post-proceedings of the 2009 EvoDeRob workshop, Studies in Computational Intelligence, Springer, accepted for publication

Svec P, Schwartz M, Thakur A, Anand D, Gupta S (2010) A simulation based framework for discovering planning logic for Unmanned Surface Vehicles. In: ASME Engineering Systems Design and Analysis Conference, Istanbul, Turkey

Thakur A, Gupta S (2010) A Computational Framework for Real-Time Unmanned Sea Surface Vehicle Motion Simulation. In: ASME 2010 International Design Engineering Technical Conferences (IDETC) & Computers and Information in Engineering Conference (CIE), Montreal, Canada

Togelius J, Burrow P, Lucas S, et al (2007) Multi-population competitive co-evolution of car racing controllers. In: IEEE Congress on Evolutionary Computation (CEC'07), September 25-28, Singapore, pp 4043–4050